Management and Search of Private Data on Storage Clouds*

Bernardo Ferreira, Henrique Domingos

Dep. de Informática, FCT/UNL
CITI Center for Informatics and Information Technologies
bernardolferreira@gmail.com, hj@fct.unl.pt

ABSTRACT

The article presents a solution for storage and management of private data kept in Internet Storage Clouds. The solution supports operations over the data kept ciphered, including reads, writes and searches based on multiple keywords and relevance classification. The approach is based on a middleware architecture supported by homomorphic encryption techniques combined with dynamic indexing mechanisms. The solution preserves conditions of privacy without need to either decipher data during operations in the cloud or transfer the data during searches. The article further describes an implementation prototype of the solution and its evaluation. The performance obtained in different implementation scenarios is analyzed and compared to a solution promoted by AMAZON S3. The evaluation shows that the solution is viable, offers more security and control for the user and does not aggravate conditions of latency and data availability.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General-Security and protection; C.2.4 [**Distributed Systems**]: Distributed applications; E.3 [**Data**]: Data encryption

General Terms

Algorithms, Reliability, Security

Keywords

Cloud Computing, Internet Storage Clouds, Security and Privacy, Homomorphic Ciphers, Search and Ranking of Data

1. Introduction

The security and dependability guarantees of data kept in storage Clouds based in solutions offered nowadays by Internet providers (or Internet Storage Cloud Solutions), are decisive criteria in the generalized adoption of those solutions. These solutions present interesting storage and remote data management characteristics, with flexibility of configuration in regard to storage space necessities in each instant and pay-per-use charging models [1]. At first glance they are advantageous solutions, from a technical or operational point of view as well as from an economical one, avoiding overloads of management and administration of the data kept by the clients themselves. The existing solutions offer a data storage service with good dependability, accessibility and availability guarantees, in ubiquitous access conditions, independent of geographical location [2]. Nonetheless, when effective and independent control conditions are required by the users over the real conditions of dependability, availability, security and privacy of the data and operations over it, the adoption of these solutions can be more problematic.

The undue access or unauthorized disclosure of private data kept in Storage Clouds has been referred as a critical problem, not only in the use case of secure data backup but also to preserve security guarantees of data accessed by online applications [3]. Such is the case for applications that manage and search medical records; access private videos or photographs; manage financial data; or access, search and consult governmental documents [4]. The Clouds dependency of third party trust bases prevents the control and complete auditing by the users of possible security vulnerabilities in the computational infrastructure (hardware/software) of the providers, allowing the data to be targeted by illicit actions or unattended operation by server technical and administrative staff. More specifically, attacks have been verified due to eventual security deficiencies or that explore physical access to the computational and communicational resources used by the providers, including network and communications equipments, software systems, or computational infrastructure devices: memory, hard drives or internal backup solutions [5].

To address the above problems in the adoption of internet storage clouds, a dependable solution is needed, conjugating three fundamental dimensions: (i) data privacy management, preserving privacy conditions during searches or other operations over it; (ii) data resilience guarantees to counter Cloud failures, data unavailability, data corruption or Cloud vendor-locking practices; (iii) scalability and performance guarantees, for the possible management of big data sets, large number of operations and possible access by multiple users.

The solutions that have been proposed in recent years are mostly based in the use of cryptographic techniques to cipher the stored data (ex. [6, 7]). In its majority they advocate the protection of data kept ciphered in the servers, being the ciphering done before the data is outsourced [8]. In order to be operated, data must be transferred to the clients that proceed to its decryption and then execute the operations. These solutions are limited when data processing at the server-side is required, for efficiency and low-latency requirements. This is the case of applications using storage clouds as remote data-storage backends or Databases as Services, provided by Cloud-based solutions. Those solutions are equally limited in applications that have to process or search big

*This work has been supported by the PhD Fellowship program of the DI-FCT-UNL, and the project FCT-MEC PTDC/EIA/113729/2009 Services for Intrusion Tolerance in Ad-Hoc Networks, DI-FCT-UNL and CITI Research Center.

SDMCMM'12, December 3-4, 2012, Montreal, Quebec, Canada.

data volumes in key-value repositories, as is the case of Cloud data processing applications.

In this article a solution is proposed that has in sight the conjugation of the three fundamental dimensions, as previously enounced. The solution addresses dependability concerns, conjugating security, reliability and availability requirements, maintaining the independent control of data by the user, promoting a trustable environment for data storage and data management on Internet provided storage clouds. The conjugation of the previous dimensions is achieved with a reference middleware architecture (explained in the section 3) for the intermediation of secure storage services and secure search of private data kept in storage Clouds. The system can use different heterogeneous Clouds, by different Internet providers, to replicate data for reliability and availability purposes. Although the data resilience and scalability dimensions are key criteria in the proposed solution, the scope of the present article focuses on the proposal of effective mechanisms for secure searching over ciphered documents stored in the Cloud (as explained in the section 2). The solution supports private data-searching with multiple keywords, ranking by their relevance and keeping conditions of data and query privacy under the full control of end users. The approach uses cryptographic schemes that explore homomorphic encryption techniques (section 4), combined with dynamic indexing mechanisms (section 5). This combination allows the preservation of privacy conditions under different implementation scenarios (section 6), without ever needing to decipher the data or proceeding to its transfer during searches.

2. Secure Searches over ciphered data

One of the most common types of searches over files or text documents is based on the use of keywords or subsets of searchable metadata. Usually the keywords and the searches are supported over the original documents in plaintext. The need to execute these searches over ciphered data raises new problems. In general, operations on the cipher data require it to possess homomorphic characteristics in relation to plaintext data. This is, apparently, a contradictory aspect, regarding to the security properties of the cryptographic algorithms themselves. This means that a good homomorphic cryptographic scheme must be able to combine the homomorphic properties required while preserving the security characteristics, in accordance to security analysis criteria as considered for conventional cryptography.

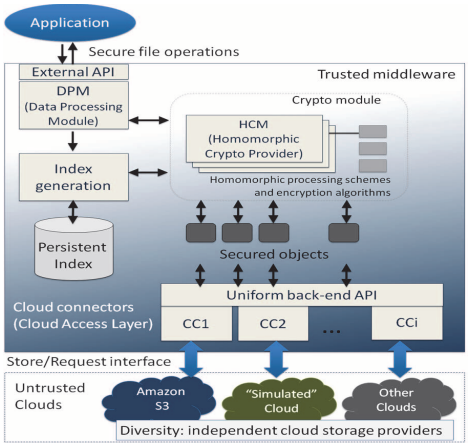

Fig. 1. Reference architecture of the envisioned solution

Different homomorphic properties may be needed for different purposes [9, 10]. A cryptographic transformation presents pure and complete homomorphism when any operation on the plaintext data can be transformed into an equivalent operation on the ciphertext data. This vision of full homomorphism does not have today a generic and practical solution based solely on pure homomorphic cryptographic algorithms. Nonetheless, partially or incomplete homomorphic encryption schemes can be addressed. A cryptographic transformation is partially homomorphic when the homomorphism is only kept for a subset of the operations, such as: a single operation or a limited group of operations (ex. additions and subtractions, multiplications and divisions, etc.).

Although some conventional encryption schemes (ex. [9, 11, 12]) allow searching over the ciphered data, exploring partially homomorphic properties, a large group of the current practical approaches only addresses Boolean searches, that is, searches that verify the (in)existence of one or more text terms. With such solutions it is not possible to capture complementary indicators, such as classification and relevance metrics, needed for multi-keyword ranked searches. In this case, the common solutions present limitations, possibly not supporting: (1) searches without previous knowledge of the ciphered data, especially when they are in big volumes; (2) searches without complete or partial data transfer to a user's trust base, where they can be decrypted during the search process; (3) searches requiring low processing latency, avoiding network traffic, also affecting the "pay-per-use models" of Cloud repositories, as usually available.

3. System model and solution architecture

The proposed solution aims at conjugating different requirements in a dependable system model and middleware architecture, for management of data privacy, authenticity, dependability, reliability and availability, using Internet Storage Clouds, under control of the users that own the data. Following this reasoning, the solution addresses the following requirements:

- Ensures the confidentiality, integrity and privacy of the data, with independent control and auditing by the users, extending these guarantees to the search operations over the ciphered data, namely supporting multi-keyword searches ranked by relevance metrics;
- Allows the integration of different and possibly heterogeneous available Internet Storage Clouds, taking advantage of the benefits they might offer, including security and reliability by exploring their diversity;
- Complements the security and privacy of data and operations with additional dependability and availability requirements, without those guarantees being dependent of each provider in particular;
- Promotes a solution that can operate under full control of the users, independently of services or guarantees of different providers and their solutions;
- Protects users from commercial or operational practices that might harm their interests (such as vendor lock-in practices [13]).

Figure 1 shows the reference architecture for the proposed solution. The architecture has in sight the use of homomorphic encryption techniques (HCM module) in conjunction with dynamic data indexing processes. As represented, different storage Clouds may be used to enhance dependability, security and availability criteria, by exploring heterogeneity and diversity.

System Model. In the system model we consider the existence of final users (who own the data) that execute applications that interact in a secure, authenticated way with the middleware, seen

as a "proxy" service that mediates the access to one more Clouds. The users possess collections of documents (typically sets of files) that can be securely transferred to the middleware. For such end, the middleware offers a secure API with the operations PUT, GET and SEARCH. The access to the middleware system might be local or remote, based on authenticated operations supported by SSL sessions. After PUT operations, the middleware system proceeds to the indexing of the data, building a searchable secure index from all the relevant keywords found in the files. The index is stored persistently and ciphered at the middleware or in one or more Clouds, depending of parameterization. The collection of ciphered documents is then stored in one or more Clouds. The documents are mutable, that is, new versions of already existing documents can be inserted, with repercussion on the metrics stored in the index. To search the collection of documents for a set of keywords, an authenticated and authorized user submits a SEARCH operation. The system can execute the SEARCH operation locally (if the index is kept at the middleware) or remotely (if the index is secured at the Cloud). The search is done by passing a set of ciphered terms, working as "trapdoors", corresponding to the ciphering of the original query keywords. In the Cloud, the terms are used directly to search on the ciphered index, returning as answer the set of relevant files and ranking information, being the answer ordered and also ciphered. Nonetheless, in all processes no Cloud server learns or has knowledge of the metrics used, the searched keywords, the data kept in the index and the documents themselves. In order to optimize the use of the network, financial cost and latency of accessing the Cloud, it is possible to parameterize the searches to return only a certain number of most relevant documents.

Adversary Model. In the previous model, the trust base for preserving conditions of dependability, availability, security and privacy is restricted to the components of the middleware system. The storage Clouds can be considered not trustable, admitting they may be subject to attacks of intrusion, as long as these attacks are carried independently on each Cloud. In the case of having the middleware implemented as a service in the Cloud, it is considered that the servers of this Cloud are not subject of attackers aiming at destroying its data, disrupting their service or *crashing* the software. It is assumed, in this case, that the HW/SW infrastructure of this Cloud is always dependable and available, executing correctly the middleware system according to specifications. Nonetheless, attacks on the communications or on the servers, aiming at breaking data privacy, are admitted.

4. Support for searching over ciphered data

We now present the three schemes conceived and implemented in the homomorphic ciphering module (HCM in the reference architecture discussed): random scheme, homomorphic scheme and linear search scheme.

Random Scheme. The most secure encryption scheme used in our solution and with best performance is the random scheme. In this scheme, two equal plaintexts originate different ciphertext with overwhelming (pseudo-random) probability, protecting against *Adaptive Chosen Plaintext* attacks (IND-CPA) [10]. Nonetheless, due to its probabilistic model, the scheme does not allow any computations over the domain of the ciphered data. These properties make it suitable for ciphering data not subject of further operations, such as the documents themselves, or even the index when it is being ciphered for persistent storage. In our solution the random scheme is built using a block symmetric ciphering algorithm, being the implementation based on AES on CBC mode, with an adequate ciphering key and random initialization vector.

Homomorphic Scheme. The solution uses an implementation of the cryptographic system Paillier [14] as homomorphic encryption scheme. This scheme allows homomorphism in relation to the addition over cipher text blocks, based on demonstrable properties where the multiplication of the ciphertexts ε(x1) and ε(x2) is equivalent to the encryption of the modular addition of their plaintexts x1 and x2. Equation (1) exemplifies the homomorphic properties of the scheme and more details can be found in [14].

$$\mathbf{E}(X_1) \,.\, \mathbf{E}(X_2) = (g^{X1} r_1{}^m) \,.\, (g^{X2} r_2{}^m) = g^{X1+X2}(r_1 r_2)^m = \mathbf{E}(X_1+X_2 \bmod m) \quad (1)$$

Linear Search Scheme. This scheme is used for ciphering textual data, allowing linear scans and searches for patterns over the encrypted data. The cipher text is calculated as authentication codes based on HMACs and a secret used as a Master Key. For instance, a client produces synthesis as HMAC over the various terms of a document, with the expression identified in (3). Afterwards, the existence of a term in the document can be verified by comparing a transformation over it (2) with the previously generated synthesis.

$$\mathbf{WordKey}(\text{word}) := \text{HMAC<MasterKey>(word)} \quad (2)$$

$$\mathbf{E}(\text{word}) := \text{Hash}(\mathbf{WordKey}(\text{word})) \;\; \text{XOR} \;\; (\text{Salt} + \text{HMAC<}\mathbf{WordKey}(\text{word})\text{> (Salt)}) \quad (3)$$

This scheme was first proposed in [9] to support linear scans and is also used in [10]. The expressions (2) and (3) represent the cryptographic functions of the scheme as presented in [10]. In the presented scheme two equal plaintexts originate different ciphertexts. Although this property is desirable from a security viewpoint, it limits the computations over the ciphertext. For instance, to support the construction of a secure index over the encrypted terms of a collection of documents, allowing the middleware architecture to be run on an uncontrolled environment such as the Cloud while preserving privacy, we need the ability to directly compare two ciphertexts for equality. Following this requirement we propose an adapted scheme that consists on removing the random vector *Salt* and HMAC synthesis from expression (3).

$$\mathbf{E}(\text{word}) := \text{Hash}(\mathbf{WordKey}(\text{word})) \quad (4)$$

The proposed scheme (4) is deterministic, as two equal plaintexts generate the same ciphertext. However, the level of security achieved can still be acceptable for the purpose in sight. More specifically, the scheme's security is inherited by the underlying security of the used hash function and by the protection of the MasterKey (which is kept protected in the user's device). The performance of the proposed scheme is also guaranteed by the hash function used. In the resulting scheme, the Hash function is kept over the HMAC function, as a way to increase the ciphertext distribution and to limit it's length.

5. Dynamic indexing

A key component in the middleware architecture is the indexing module for support of secure multi-keyword searches with ranking over the secure data ciphered and stored at the Cloud. To speedup queries an index is built before hand and then managed through dynamic insertions and deletions of documents. We now present the indexing mechanisms that allow these operations.

Indexing and Scoring. An index for text data can be efficiently represented as a set of inverted lists [15]. Figure 2 shows an example implemented over a *HashMap*. Summarizing, the index is composed by a dictionary that maps each searchable term (that appears in one or more documents) to a list of integer pairs, known as a *PostingList* [15]. Each pair in this list represents a document where the term appears and a score for the corresponding term/document relation that quantifies an indicator of relevance. In our case, we store as score the direct frequency, that is, how many times the term appears in the document. As text is characterized as being sparse data, meaning that a term generally only appears in a fraction of all the documents, each *PostingList* contains only the required size in each case (zero frequency documents are not stored).

As direct frequency alone may not truly represent the value of a document when represented as part of a whole repository, it's necessary to collect additional more general metrics to access the effective relevance in a query. Among them we can count the size of each document, the average size of the collection and document frequency (the number of documents that a term appears on). When a search is done, these statistics are grouped in a scoring function that allows us to assess the final ranking of the query. In our solution we used, from the various functions published to this date, the Okapi BM25 scoring function [16], as it is a probabilistic function that conjugates the various metrics described and as it is one of the most analyzed and used in the Information Retrieval community. If a query possesses multiple terms, all the corresponding PostingLists are fetched from the index, transformed into final scores using the function and then merged into the final result.

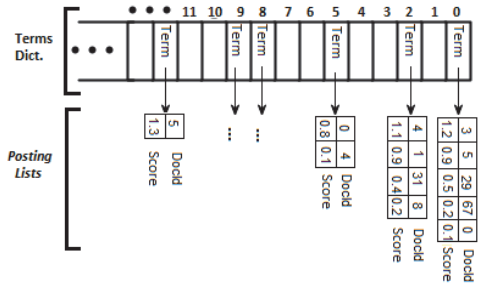

Fig. 2. Inverted List Index implemented on *HashMap*

The construction of the index is done using the SPIMI algorithm [17], chosen for its performance, scalability and low memory requirements. As this algorithm does frequent searches over the PostingLists as it processes the documents, we chose to implement them also as HashMaps. After the index has been built, and in case of seldom dynamic insertions, the maps can be converted into arrays in order to compress the size of the index in memory. On the other hand, if the index does not fit entirely in memory, during its construction we use the support of secondary memory (as described in [17]) and in the end we use a technique called *ChampionLists* [15]. This technique consists in keeping the whole index in persistent storage and leaving in memory only the top scoring documents for each term, ordered by their relevance. This way searches can be done without the overhead of accessing hard disks. With this technique we lose some precision, nonetheless it is not expected to be substantial nor relevant for the final result.

Dynamic Insertions and Removals. The indexing module must be able to deal with document insertions and removals without compromising the searching functionality and its precision. Dynamic changes can occur in two circumstances: when the index still fits in memory and when it has already been written to disk. In case it is still in memory, the update is trivial. New documents are processed as usual and resulting *PostingsLists* are merged with the index; removed documents are deleted from the index, as well as its references and metrics. In case the index is stored in disc, the possible retrieval to memory of various partitions of the index in order to update them becomes too expensive. As such, the solution consists in creating an auxiliary index in memory that will hold the new *PostingLists*, as well as a list with identifiers of the removed documents. These structures will exist in memory in parallel with the *ChampionLists* referenced previously. Searches are done having in consideration both indexes and filtering the removed documents from the results. Periodically the auxiliary index is merged with the full index in disc, documents referenced in the removal list are effectively deleted from the Cloud (and from the index) and the *ChampionLists* are recalculated. This "re-indexing" can, nonetheless, be carried in background without interrupting executing searches.

6. Discussion and implementation of different case studies

The integration of the middleware components for indexing and data security is possible in different ways, providing flexibility for different purposes, as described next. The presented implementation scenarios focus on possible implementations of the solution, in a single user perspective. The generalization of the solutions to multi-user environments is in our on-going research work agenda.

Middleware operating in the user's device. In this case the index is built and kept in the middleware, operating locally in the use's device. Documents are sent to the Cloud, ciphered with the Random scheme and identified by a randomly created identifier. As the user's device is assumed to be trustable, the index does not need to be ciphered. This scenario, although simplistic, is the one that achieves better security guarantees and performance. However it is also the one that requires more computational power in the client.

Middleware as a proxy service in a local network. In this scenario the Cloud is used for storing the ciphered documents but the index is stored ciphered in a local proxy server. The ciphering of the index is done with the Random scheme, forcing the client device to decipher and merge PostingLists fetched as result of a search in order to obtain the final ranked results. As alternative we can use the Homomorphic scheme to cipher the scores stored in the index, increasing overhead of indexing and querying but freeing the client from any computation during searches. The construction of the index still has to be done on the client, as it is necessary a secure environment to process the documents.

Middleware as a service in the Cloud. In this case the index is built and stored ciphered in the Cloud, allowing a soft-state in the client who only has to store the necessary cryptographic keys. To this end we use the Linear Search scheme, ciphering each searchable term in every document and then sending the ciphers to the Cloud. With this scheme, the ciphers can be compared without knowledge of their actual contents. This mechanism allows the middleware running in the Cloud to collect the necessary relevance metrics, thus building the index in an uncontrolled environment while preserving privacy and confidentiality requirements. There is a forced relaxation of the level of security,

as the collected statistics will be revealed during the indexing process. Despite this, the problem can be mitigated if the index is ciphered after construction (with the Random or Homomorphic scheme), allowing secure searches to be done without revealing any further information.

7. Implementation and evaluation

In order to experimentally validate the conceived solution, a middleware system prototype was implemented (programmed in Java) and different load tests were carried. The presented preliminary evaluation is focused primarily on performance criteria and latency considerations. We will extend the observations of other important dimensions, such as reliability, resilience and scalability conditions, as future-work.

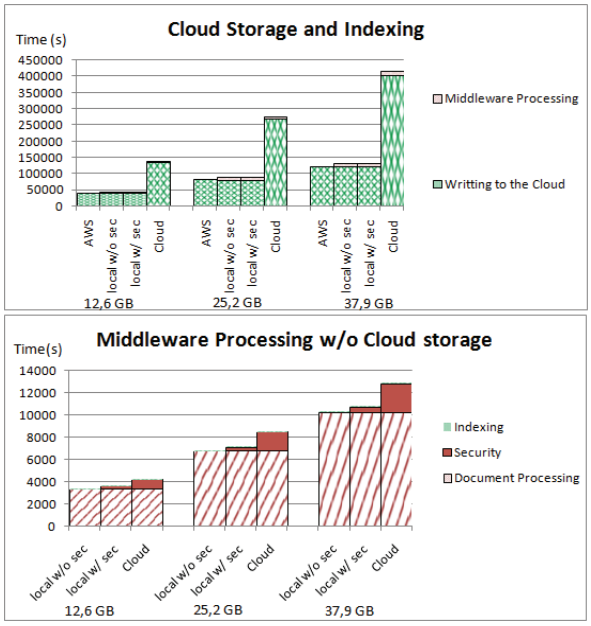

Fig. 3. Performance of writing (3(a)) and indexing (3(b)) in the different scenarios

The experiments and results we now present were obtained using a *JVM OpenJDK 6*[1], with 6 GB of initially reserved and limited memory, *Concurrent Garbage Collector*[2] activated and executing in a PC with *Intel Core i3 3.4Ghz* processor. The chosen Cloud provider was Amazon[3], Ireland data-center (Amazon *S3* as storage service, Amazon EC2 as computational Cloud service, configured with a Large Instance). The connection to the Cloud was limited to 30 Mb/s for downloads and 3 Mb/s for upload. The dataset used was the collection of all English Wikipedia pages[4], as of May 2012, with a total uncompressed size of 37,9 GB.

The executed tests are divided in two groups. First we tested the performance of writing, storing and indexing the collection, in the different implementation scenarios described in the previous section and comparing with the direct storage of the documents in the Cloud (using the secure solution from Amazon for *Client-Side Encryption*[5]). The second group of tests compared the performance of searching in the different implementation and operation scenarios.

Figure 3 shows two related graphs. Graph 3(a) represents the times of the writing process as a whole, comparing the secure solution from Amazon with the middleware's performance in the different scenarios.

Graphic 3(b) shows the overhead of the middleware processing alone, without Cloud writing, and divided by its main processes (document processing, only done once for each document; indexing or re-indexing of the relevance metrics collected during document processing; and security). The results show that the overhead introduced by the indexing mechanism is minimal when compared to the cost of sending the same collection of documents to the Cloud. On the other hand, joining "conventional" security to the index built locally (or in a trusted proxy) also adds little latency to the process. As for the process of indexing at the Cloud, overhead is slightly increased in the security part as due to the necessity of ciphering each word of each document independently with the Linear Search Scheme. Also, writing to the Cloud in this scenario is considerably slower, as we have to send the collection of documents twice to the Cloud: once ciphered with the Random scheme, for storage, and a second time ciphered with the Line Search Scheme, for indexing. This necessity comes from the irreversibility of the last.

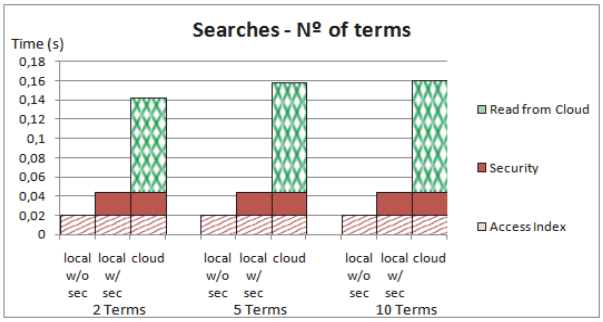

Fig. 4. Search performance in the different scenarios

Figure 4 shows the results of the second group of tests, using the 37,9 GB collection of English Wikipedia documents. Searches with or without security are very fast and are always done under the 100 milliseconds mark. Fetching search results from the Cloud, in the respective scenario, slightly increases latency due to network traffic. Nonetheless, even in these cases we achieve search times bellow 500 milliseconds, which demonstrate the viability of the search mechanism (always inferior to 1 second, the threshold for online applications). Results also show that performance is preserved as query size increases.

8. Related work

In the last years some articles have been published on the topic of trustable and secure data storage in the Cloud. Among these stands out DepSky [18], that is based on a solution where the requirements of security and dependability (in a dependable system vision) are addressed by a model and architecture of "Cloud of Clouds". The solution offers resilience, high availability, confidentiality and data integrity, as well as defense against practices of *Cloud Vendor Lock-in* and uses symmetric cryptography based on secret sharing schemes. For that purpose, the system uses techniques for byzantine fault tolerance in the Clouds, with data replication and the use of byzantine quorum protocols. In our solution we foresee, as a future work direction, the integration of this type of mechanisms in the middleware services, in addition to the homomorphic ciphering and dynamic indexing modules, joining the potential of rich searches over the ciphered data with a cloud of Clouds approach under full control of the user.

Another relevant work is the iDataGuard system [19]. This approach addresses confidentiality and data integrity, as well as searches over the ciphered data. The system uses schemes inspired

[1] http://openjdk.java.net/projects/jdk6/
[2] http://docs.oracle.com/javase/6/docs/technotes/guides/vm/cms-6.html
[3] http://aws.amazon.com/
[4] http://en.wikipedia.org/wiki/Wikipedia:Database_download
[5] http://aws.amazon.com/articles/2850096021478074

in symmetric cryptography and indexes built for supporting Boolean searches over data stored in not trusted repositories (without ranks or relevance classification). The system does not allow the flexibility of architecture presented in our work, executing only as processing component on the client side. Furthermore, our conceived system allows implementing the same solution offered by the iDataGuard system, enriching the type of available searches that can be done over the ciphered data.

One of the inspiring works in the area of secure systems for the Cloud is CryptDB [10]. This system combines different ciphering schemes, including homomorphic ciphers, in order to allow secure and private searches over private data in SQL databases based on the Cloud (inspiring a direction of "Secure Relational Database as a Service in the Cloud"). The system addresses secure SQL queries, while our work focuses more on text document repositories and key-value stores without pre-defined schema. The homomorphic schemes presented in the work inspired our solution, while also allowing the exploration, in the future, of other forms of searching, including searches for metadata associated with the documents that may contain private multimedia information stored in repositories based on Cloud solutions.

9. Conclusions

The article presents a solution that has in sight the conjugation of dependability, availability, security and privacy requirements of data stored in Internet Storage Clouds. The solution is addressed as a middleware system for intermediation of secure storage services for private data in storage Clouds. The presented system supports the management and storage of private data, under full control of the user, and allows searching over the ciphered data, independently of different Clouds that may be used. A relevant contribution of the proposed solution focuses on the support of secure searches over the data, addressing a solution for multi-keyword searches, with criteria for classification of documents. The solution achieved allows the use of effective mechanisms for searching over the private documents with multiple keywords and accessing the ciphered information in the Cloud, based on scoring/ranking operations on the relevance of the data. During the search operations, privacy conditions are preserved under full control of the users. The presented approach uses cryptographic schemes that explore homomorphic encryption techniques combined with dynamic indexing mechanisms. The implementation of the proposed system and its evaluation shows that the solution is viable, offers more security and greater user control (comparing to a solution promoted by Amazon AWS) and does not aggravate conditions of access latency and data availability.

References

1. P. Mell and T. Grance, "Draft nist working definition of *Cloud* computing," Referenced on Jan. 23rd, 2010 Online at http://csrc.nist.gov/groups/SNS/*Cloud*-computing/index.html, 2010.
2. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A View of *Cloud* Computing," Communications of the ACM 53 (4) 50-58, April 2010.
3. Privacy Rights Clearinghouse. Chronology of data breaches. http://www.privacyrights.org/data-breach.
4. S. Kamara and K. Lauter, "Cryptographic *Cloud* storage," in Proceedings of Financial Cryptography: Workshop on Real-Life Cryptographic Protocols and Standardization 2010, January 2010.
5. J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember: Cold boot attacks on encryption keys. In Proceedings of the 17th Usenix Security Symposium, San Jose, CA, July–August 2008.
6. J. Li, M. Krohn, D. Mazi`eres, and D. Shasha. Secure untrusted data repository (SUNDR). In Proceedings of the 6th Symposium on Operating Systems Design and Implementation, pages 91–106, San Francisco, CA, December 2004.
7. A. J. Feldman, W. P. Zeller, M. J. Freedman, and E. W. Felten. SPORC: Group collaboration using untrusted *Cloud* resources. In Proceedings of the 9th Symposium on Operating Systems Design and Implementation, Vancouver, Canada, October 2010.
8. *Cloud* Security Alliance, "Security guidance for critical areas of focus in *Cloud* computing," 2009, http://www.*Cloud*securityalliance.org.
9. D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in *Proc. of IEEE Symposium on Security and Privacy'00*, 2000.
10. R. Popa, C. Redfield, N. Zeldovich, H. Balakrishnan. *CryptDB: Protecting Confidentiality with Encrypted Query Processing*. SOSP '11, October 23–26, 2011, Cascais, Portugal.
11. D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. of EUROCRYP'04, volume 3027 of LNCS*. Springer, 2004.
12. R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: improved definitions and efficient constructions," in *Proc. of ACM CCS'06*, 2006.
13. H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon. RACS: A case for *Cloud* storage diversity. Proc. of the 1st ACM Symposium on *Cloud* Computing, pages 229–240, June 2010.
14. P. Paillier. *Public-key cryptosystems based on composite degree residuosity classes*. In Proceedings of the 18th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT), Prague, Czech Republic, May 1999.
15. C. Manning, P. Raghavan, H. Schütze. "An Introduction to Information Retrieval", Cambridge University Press, 2009
16. Spärck Jones, Karen, S.Walker, and Stephen E. Robertson. 2000. A probabilistic model of information retrieval: Development and comparative experiments. *IP&M* 36(6): 779–808, 809–840.
17. Heinz, Steffen, and Justin Zobel. 2003. Efficient single-pass index construction for text databases. *JASIST* 54(8):713–729. DOI: dx.doi.org/10.1002/asi.10268
18. A. Bessani, M. Correia, B. Quaresma, F. André, P. Sousa. *DEPSKY: Dependable and Secure Storage in a Cloud-of-Clouds*. EuroSys'11, April 10–13, 2011, Salzburg, Austria
19. R. Chandra, R. Gamboni, S. Mehrotra, K. Seamons, N. Venkatasubramanian. *iDataGuard: An Interoperable Security Middleware for Untrusted Internet Data Storage*. Middleware'08 Companion, December 1-5, 2008, Leuven, Belgium.